Komputerowe modelowanie zjawisk rynkowych Documentation

Wydanie 0.1

Łukasz Machura

October 14 2012

Spis treści

Zajęcia dla 1 roku studiów magisterskich ekonofizyki na Uniwersytecie Śląskim w semestrze zimowym roku akademickiego 2012/2013.

Autor Łukasz Machura, ZFT, Uniwersytet Śląski

Wersja 0.1 (Ekonofizyka UŚ, semestr zimowy 2012)

ROZDZIAŁ 1

Wstęp

Wymagania teoretyczne

- 13. Łukaszewski, J. Sładkowski, Wybrane zagadnienia analizy rynków finansowych
- 13. Łukaszewski, J. Sładkowski, Wprowadzenie do funkcjonowania rynków finansowych
- 10. Łuczka, Procesy i zjawiska losowe

Wymagania techniczne

- podstawy programowania (Sage/Python lub Matlab/Octave lub C/C++ lub cokolwiek innego...)

Uwagi

- Jako, że zajmujemy sie metodami komputerowymi, warto ten podręcznik czytać przy komputerze, jednocześnie powtarzając prezentowane obliczenia samodzielnie. Większość zaprezentowanego kodu dostępna jest w projekcie kmzj, zalecam jednak wpisywanie większości algorytmów samodzielnie. Nauka poprzez palce jest bardzo ważna, pozwala nabyć niezbędnych umiejętności pracy przy (na?) komputerze. Zwykłe "copy-paste" nie nauczy nikogo wiele. Dodatkowo zachęcam do dowolnych zmian w kodach prezentowanych w tym kursie - do optymalizacji, skracania, przyspieszania itp. Nie zawsze pisałem ten kod z myślą o zastosowaniach końcowych. W większości przypaków ma on być szkoleniowy i intuicyjny (jeżeli dla kogoś Python jest intuicyjny).
- W tym kursie do numerycznych realizacji używać będziemy pakietu Sage. Jest to dostępny dla każdego program typu open-source bazujacy na języku Python.

Liczby Losowe

2.1 Liczby losowe i pseudolosowe

Intuicyjnie dość dobrze rozumiemy co oznacza termin *liczba losowa*. Każdy z nas choć raz w życiu podrzucił monetę do góry po to, by "ślepy los" zdecydował za niego o jakimś wyborze (jeżeli w ten sposób zdecydowaliście o wyborze studiów, to szczerze mówiąc - gratuluję). Oczywiście na monecie nie ma żadnych liczb, ale można sobie potraktować reszkę (R) jako 0 a orła (O) jako 1 (co bardzo dobrze reprezentuje fałsz i prawdę lub niemożliwe i pewne zdarzenie w teorii prawdopodobieństwa). Teraz już możemy sobie podrzucać monetę i na kartce papieru zapisywać kolejne wylosowane (wyrzucone) przez nas liczby

```
0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
```

co odpowiada oczywiście wyrzuceniu kolejno

```
R, O, R, R, R, R, O, R, O, O, R, R, O, R, O.
```

W naszym przypadku zapiszemy sobie te liczby od razu do listy w notatniku Sage.

```
rzuty = [0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1]
```

Mamy teraz je dostępne pod zmienną `rzuty`. Do prostych zagadnień, gdzie potrzebne jest nam kilka, czy nawet kilkanaście takich liczb, bez problemu możemy poradzić sobie rzucając monetą. Jeżeli potrzebujemy zadecydować o wyborze pomiędzy trzema możliwościami możemy użyć sześciennej kości do gry i przykładowo wybrać wynik poprzez działanie modulo 3 (mod3). Tym razem dostaniemy trzy możliwe liczby `0, 1, 2`

```
# rzuty kością
5, 3, 6, 5, 6, 6, 5, 3, 2, 4, 3, 1, 1, 6, 1,

# modulo 3
2, 0, 0, 2, 0, 0, 2, 0, 2, 1, 0, 1, 1, 0, 1.
```

Sytuacja zrobi się jednak nieco bardziej skomplikowana, gdy będziemy potrzebować tysiąc, milion czy bilion takich liczb. Jeżeli nawet grupa 10 studentów była by w stanie wyrzucić monetą tysiąc losowych zer i jedynek w pół godziny (włączając w to zapisywanie w liście Sage lub nawet na kartce papieru) to uzyskanie miliona liczb jest praktycznie nie do zrobienia w ten sposób. problem pojawia się też w momencie, gdy chcielibyśmy mieć liczby naturalne z zakresu np: $0 - 10$, czy w końcu losowe liczby zmiennoprzecinkowe. Metody chałupnicze w tym momencie się kończą.

Z pomocą może przyjść nam komputer (w końcu ten wykład jest o metodach komputerowych). Obecnie znakomita większość języków programowania (przynajmniej tych realnie wykorzystywanych [1]) posiada w swoich standardo-

[1] Generator liczb pseudolosowych można napisać nawet dla tak egzotycznych języków jak Brainf*ck.

wych bibliotekach funkcje (metody, klasy) umożliwiające wygenerowanie liczby (pseudo)losowej z przedziału `[0,1)` lub też `[0,RAND_MAX]`, gdzie ów `RAND_MAX` to stała zależna od architektury komputera, kompilatora i bibliotek.

Uwaga Jako, że programy/języki komputerowe zawsze generują liczby pseudolosowe, od tej pory mówiąc o liczbach losowych generowanych w Sage/Python czy innych językach, będziemy zawsze mieć na myśli liczby pseudolosowe.

W Sage liczby losowe uzyskuje się poprzez funkcję `random()`. Zwraca ona liczbę losową z przedziału `[0.0, 1.0)`. Wykorzysując proste wyrażenie listowe możemy przypisać do listy N liczb losowych.

```
N = 1000
lista = [random() for i in xrange(N)]
```

Inna funkcja `randint(a, b)`, zwraca liczby całkowite z przedziału `[a,b]`. Czyli symulacja rzutu monetą może być zrealizowana poprzez

```
rzut_moneta = [randint(0,1) for i in xrange(N)]
```

Zadanie Zamodeluj w Sage rzut kością. Wygeneruj listę 1000 liczb odzwierciedlających 1000 rzutów symetryczną sześcienną kością do gry. Wynik zapisz w zmiennej `rzut_kostka`.

Matematycznie rzecz biorąc liczbę losową można utożsamić z wartością jaką przybiera pewna zmienna losowa ξ. Możemy napisać, że dla procesu jakim jest rzut kością zmienna losowa ξ może przybierać wartości `0` lub `1`. Matematyczne konsekwencje poznaliście już na wykładzie Procesy i zjawiska losowe, tutaj zajmiemy się znacznie szerzej generowaniem liczb losowych i wykorzystaniem ich właśnie do realizacji procesów losowych, ze szczególnym uwzględnieniem zastosowania dla rynków finansowych, czy w ogólności w modelach ekonomicznych.

No koniec tego rozdziału musimy sobie powiedzieć jasno: program komputerowy bazujący na deterministycznym generatorze liczb losowych może wygenerować tylko i wyłącznie liczby pseudolosowe, czyli takie, które tylko imitują prawdziwe liczby czysto losowe. Te ostatnie osiągalne są tylko procesie rzeczywistym. Możemy jednak za pomocą takich generatorów uzyskać ciąg liczb (bitów), który pod pewnymi względami będzie nierozróżnialny od ciągu uzyskanego z prawdziwie losowego źródła (np: z rzutu rzeczywistą kością).

2.2 Generatory liczb

Generator liczb losowych (RNG, z ang. random number generator) lub nieco bardziej ściśle *generator zdarzeń losowych* (REG, z ang. random event generator) to układ produkujący losowy ciąg elementów binarnych (bitów) najczęściej ułożony w postaci szeregu liczb losowych. Z punktu widzenia sposobu generowania liczb losowych wyróżniamy generatory sprzętowe (fizyczne, rzeczywiste) i programowe.

2.2.1 Generatory sprzętowe

TRNG (z ang. True RNG) - działające na zasadzie obrazowania właściwości i parametrów fizycznego procesu stochastycznego. Może to być ów rzut kością, monetą, wybieranie karty z talii kart itp. Wykorzystywać można też: efekt fotoelektryczny, szum termiczny, szum śrutowy, proces zaniku radioaktywnego...

2.2.2 Generatory programowe

PRNG, (z ang. Pseudo RNG) - działające na zasadzie deterministycznego obliczania ciągu liczb, które wyglądają jak liczby losowe. Algorytmy realizujące PRNG istnieją już ponad pół wieku i są obecnie zaimplementowane dla większości języków programowania. Na podstawie początkowej wartości nazywanej ziarnem czy zarodkiem (z ang. seed) oblicza kolejne wartości. Obie prezentowane funkcje Sage (`random` i `randint`) korzystają właśnie z jednego z takich algorytmów, zwanego Mersenne Twister. Jest to obecnie chyba najbardziej popularny algorytm opracowany w 1997 roku. Np. Matlab/GNU Octave też wykorzystuje ten algorytm. Jest on stosunkowo skomplikowany i może

być trudny do realizacji, dlatego też omówimy sobie dużo prostszy, liniowy generator i omówimy jego zalety i (przede wszystkim) wady.

Liniowy generator kongruencyjny

LCG (linear congruential generator) wyznaczony jest przez metodę rekurencyjną

$$X_{n+1} = (aX_n + c) \mod m.$$

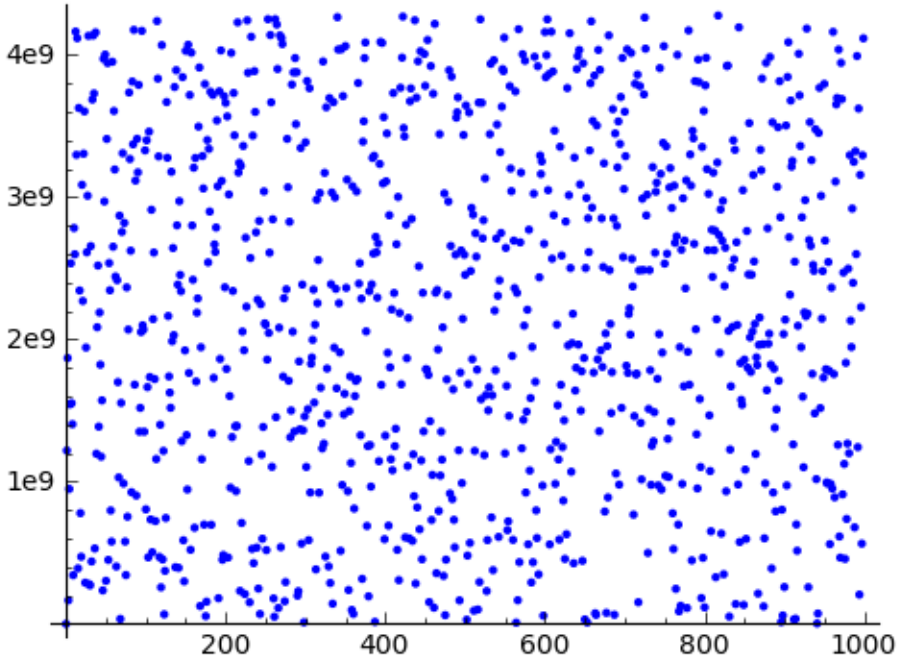
Stan początkowy to wartość ziarna. Nie jest on zbytnio bezpieczny - istnieją techniki identyfikacji parametrów modelu na podstawie obserwacji wyników. Dla niektórych parametrów jest prawie losowy a dla innych dość szybko staje się okresowy. Niemniej jednak implementacja w Sage nie powinna nastręczać zbytnich problemów.

```
def mylcg(x, a=1664525, b=1013904223, m=2**32):
  return mod(a*x+b,m)
```

Możemy teraz wygenerować N liczb używając LCG i zmagazynowac je w pythonowskiej liście.

```
def get_from_LCG(n=1, seed=123):
  ret = [seed]
  for i in xrange(n-1):
    ret.append(mylcg(ret[i]))
  return ret

lcg_list = get_from_LCG(N)
```



Rysunek 2.1: 1000 liczb losowych wygenerowanych generatorem liniowym LCG

Ala ma kota

2.3 Rozkłady

N(0,1)

2.4 Zadania

Zad Ile jest 2 + 2

Indices and tables

- *genindex*
- *modindex*
- *search*